

# BASH (Bourne Again Shell)

---

Guillaume Chanel - [guillaume.chanel@hesge.ch](mailto:guillaume.chanel@hesge.ch)

Florent Glück - [florent.gluck@hesge.ch](mailto:florent.gluck@hesge.ch)

March 18, 2025

ISC - HEPIA

Qu'est-ce qu'un shell ?

---

Pourquoi utiliser une interface texte en 2025 ?

- Interface textuelle, ligne de commande
- presque toutes les fonctionnalités d'Unix sont accessibles
- Petits outils simple que l'on peut composer
- Possibilité d'écrire des **scripts**

# Terminal & Pseudo-Terminal



Terminal



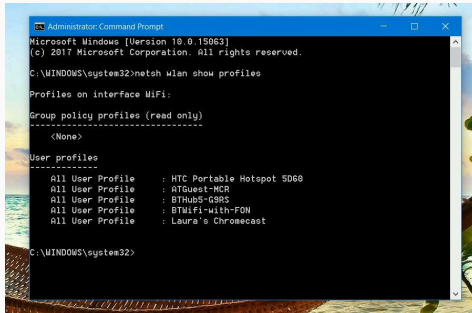
VT220

```
henni@henni:~  
henni 6941 6911 0 12:47 ? 00:00:00 kde-window-decorator -session 10d1656e00012070574670  
henni 6944 1 0 12:47 ? 00:00:11 yakuake -session 10d1656e00012071688080000085160829  
henni 6947 6911 0 12:47 ? 00:00:00 konqueror [kdeinit] --preload  
henni 6952 1 0 12:47 ? 00:00:00 klipper [kdeinit]  
henni 6954 1 0 12:47 ? 00:00:00 kalarnd --autostart  
henni 6959 1 0 12:47 ? 00:00:00 knotify [kdeinit]  
henni 6967 6911 0 12:47 ? 00:00:00 konqueror [kdeinit] --preload  
henni 6969 6911 0 12:47 ? 00:00:00 ksysrcmd /usr/bin/thunderbird-bin  
henni 6970 6969 0 12:47 ? 00:00:00 /bin/bash /usr/libexec/mozilla-launcher  
henni 6980 6970 0 12:47 ? 00:00:29 /opt/thunderbird/thunderbird-bin  
henni 6984 1 0 12:47 ? 00:00:00 /usr/libexec/gconfd-2 19  
henni 7083 1 0 12:52 ? 00:00:00 /usr/bin/gpg-agent --sh --daemon --write-env-file /hom  
henni 7671 6911 0 16:15 ? 00:00:00 kio file [kdeinit] file /home/henni/.kde3.5/socket-hen  
henni 7677 6918 0 16:15 ? 00:00:01 konqueror /home/henni  
henni 7711 1 1 16:16 ? 00:01:06 amarokapp  
henni 7723 7711 0 16:16 ? 00:00:00 ruby /usr/share/apps/amarok/scripts/score_default/scor  
henni 7724 7711 0 16:16 ? 00:00:00 ruby /usr/share/apps/amarok/scripts/lyrics_lyrc/lyrics  
henni 7725 7711 0 16:16 ? 00:00:00 /usr/bin/python /home/henni/.kde3.5/share/apps/amarok/  
henni 18419 1 0 14:04 ? 00:00:00 kio userver [kdeinit]  
henni 19409 6911 0 17:20 ? 00:00:00 kio http [kdeinit] http /home/henni/.kde3.5/socket-hen  
henni 19716 6944 0 17:28 pts/1 00:00:00 /bin/bash  
henni 20046 6918 0 17:37 ? 00:00:00 /usr/kde/current/bin/ksnapshot  
henni 20050 6911 0 17:37 ? 00:00:00 kio file [kdeinit] file /home/henni/.kde3.5/socket-hen  
henni 20077 19716 2 17:37 pts/1 00:00:00 xterm  
henni 20079 20077 4 17:37 pts/2 00:00:00 bash  
henni 20090 20079 0 17:37 pts/2 00:00:00 ps -ef  
henni@henni ~ $
```

Pseudo-terminal

# Autres interfaces en ligne de commande

## Windows



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>netsh wlan show profiles

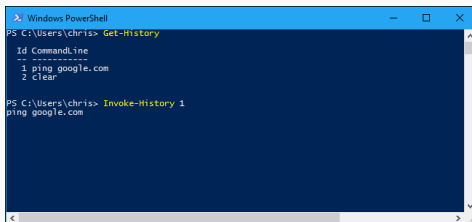
Profiles on interface WiFi:

Group policy profiles (read only)
-----
<None>

User profiles
-----
All User Profile : HTC Portable Hotspot 5060
All User Profile : ATQuest-WCR
All User Profile : BTHub5-698S
All User Profile : BTWifi1-with-FON
All User Profile : Laura's Chromecast

C:\WINDOWS\system32>
```

CMD



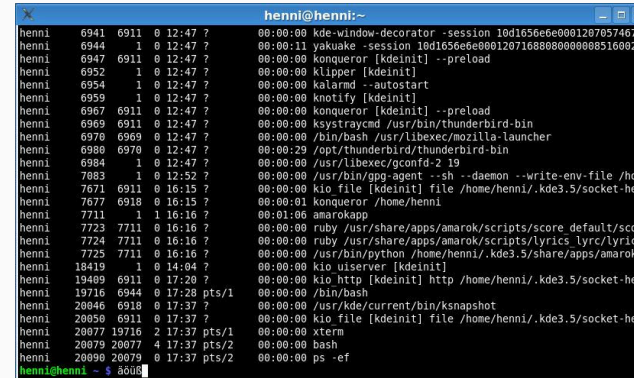
```
Windows PowerShell
PS C:\Users\chris> Get-History

Id Commandline
-----
1 ping google.com
2 clear

PS C:\Users\chris> Invoke-History 1
ping google.com
```

PowerShell

## Linux / Mac / UNIX



```
henni@henni:~
henni 6941 6911 0 12:47 ?
henni 6944 1 0 12:47 ?
henni 6947 6911 0 12:47 ?
henni 6952 1 0 12:47 ?
henni 6954 1 0 12:47 ?
henni 6959 1 0 12:47 ?
henni 6967 6911 0 12:47 ?
henni 6969 6911 0 12:47 ?
henni 6970 6969 0 12:47 ?
henni 6980 6970 0 12:47 ?
henni 6984 1 0 12:47 ?
henni 7083 1 0 12:52 ?
henni 7671 6911 0 16:15 ?
henni 7677 6918 0 16:15 ?
henni 7711 1 1 16:16 ?
henni 7723 7711 0 16:16 ?
henni 7724 7711 0 16:16 ?
henni 7725 7711 0 16:16 ?
henni 18419 1 0 14:04 ?
henni 19409 6911 0 17:20 ?
henni 19716 6944 0 17:28 pts/1
henni 20046 6918 0 17:37 ?
henni 20050 6911 0 17:37 ?
henni 20077 19716 2 17:37 pts/1
henni 20079 20077 4 17:37 pts/2
henni 20090 20079 0 17:37 pts/2
henni@henni ~ $
```

sh, **bash**, zsh, dash, csh, fish, etc.

# Préparation de l'environnement de travail

- Ouvrez un pseudo-terminal (si pas déjà ouvert)



# Préparation de l'environnement de travail

- Ouvrez un pseudo-terminal (si pas déjà ouvert)

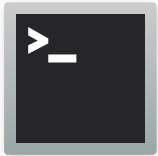


- Qu'observez vous ?



# Préparation de l'environnement de travail

- Ouvrez un pseudo-terminal (si pas déjà ouvert)



- Qu'observez vous ?
- Tapez la commande suivante :

```
$ git clone https://gitlab.unige.ch/outils-info/bash.git
```

# Manuel



# Accéder au manuel (man)

- Un manuel très complet est présent sur les systèmes UNIX
- Commande :

```
man [section] <sujet>
```

- Pour sortir du manuel, appuyer sur la touche **q**
- Pour chercher le texte **bla**, taper **/bla** (puis presser ENTER)

# Manuel : exemples

```
$ man pwd
```

# Manuel : exemples

```
$ man pwd
```

```
$ man 3 getcwd
```

# Manuel : exemples

```
$ man pwd
```

```
$ man 3 getcwd
```

```
$ man man
```

# Manuel : exemples

```
$ man pwd
```

```
$ man 3 getcwd
```

```
$ man man
```

```
$ apropos sujet # permet de lister les pages d'un sujet
```

# Sections du manuel sur GNU/Linux

1 Commandes générales

---

2 Appels systèmes

---

3 Librairie C

---

4 Fichiers spéciaux

---

5 Formats de fichiers et conventions

---

6 Jeux

---

7 Divers

---

8 Commandes d'administration et démons





# RTFM





# Se localiser et se déplacer

---

# Chemin d'accès

Les conventions suivantes (POSIX) s'appliquent :

- La racine est représentée par `/`
- Les répertoires sont séparés par `/`
- Tout chemin qui ne commence par `/` est relatif au répertoire courant
- Le répertoire courant est symbolisé par `.`
- Le répertoire parent est symbolisé par `..`
- Le répertoire *home* est symbolisé par `~`

## Exercice : `pwd`, `ls`, `cd`

1. Utilisez `pwd` pour afficher le répertoire dans lequel vous vous trouvez
2. Utilisez `cd` pour vous déplacer dans le répertoire `/tmp`
3. Vérifiez que vous êtes bien dans le répertoire `/tmp`
4. Utilisez `ls` pour lister le contenu du répertoire courant
5. Utilisez `cd` pour revenir dans votre répertoire *home*
  - trouvez deux manières différentes de réaliser ceci
6. Allez dans le répertoire `bash` (celui créé par la commande `git`)
7. Listez le contenu du répertoire `/tmp` depuis le répertoire `bash`

# Exercice : chemin d'accès

1. Quel est le résultat des commandes ci-dessous ?

```
$ cd ~/../../../../bash
```

# Exercice : chemin d'accès

1. Quel est le résultat des commandes ci-dessous ?

```
$ cd ~/./././././bash
```

```
$ ls ~/./bash/./bash/././.
```



# Exercice : chemin d'accès

1. Quel est le résultat des commandes ci-dessous ?

```
$ cd ~/../../../../bash
```

```
$ ls ~/./bash/../../bash/../../.
```

2. Comment lister le contenu de votre répertoire *home* depuis n'importe quel répertoire ? Trouvez deux moyens de le faire, puis vérifiez vos méthodes.

# Nommer un ensemble de fichiers (1/2)

Les noms de fichiers suivent les règles suivantes :

- Maximum 255 caractères (⚠️ dépend du système de fichiers utilisé !)
- Tous les caractères sont autorisés, mais les suivants sont déconseillés :
  - espaces
  - \* ? /
- Si le nom du fichier commence par un point . alors il est **caché**

# Nommer un ensemble de fichiers (2/2)

On peut référencer plusieurs fichiers en utilisant les symboles suivants:

- \* remplace zéro, un ou plusieurs caractères
- ? un seul caractère

## Exercice : fichiers cachés et sélection (1/2)

1. Avec le manuel trouvez l'option de `ls` qui permet de lister les fichiers cachés

## Exercice : fichiers cachés et sélection (1/2)

1. Avec le manuel trouvez l'option de `ls` qui permet de lister les fichiers cachés
2. Utilisez cette option pour lister également les fichiers cachés de votre répertoire *home*

## Exercice : fichiers cachés et sélection (2/2)

1. Rendez vous dans le répertoire `bash/ressources`

## Exercice : fichiers cachés et sélection (2/2)

1. Rendez vous dans le répertoire `bash/ressources`
2. Listez dans le répertoire `images` toutes les images au format JPEG (jpg)

## Exercice : fichiers cachés et sélection (2/2)

1. Rendez vous dans le répertoire `bash/ressources`
2. Listez dans le répertoire `images` toutes les images au format JPEG (jpg)
3. Lister toutes les images d'oiseaux



## Exercice : fichiers cachés et sélection (2/2)

1. Rendez vous dans le répertoire `bash/ressources`
2. Listez dans le répertoire `images` toutes les images au format JPEG (jpg)
3. Lister toutes les images d'oiseaux
4. Lister toutes les images d'oiseaux au format PNG (png)

## Exercice : fichiers cachés et sélection (2/2)

1. Rendez vous dans le répertoire `bash/ressources`
2. Listez dans le répertoire `images` toutes les images au format JPEG (jpg)
3. Lister toutes les images d'oiseaux
4. Lister toutes les images d'oiseaux au format PNG (png)
5. Lister les images de chat portant un numéro entre 90 et 99 (inclus)

# Utilisateurs, groupes et droits

---

# Informations sur les fichiers/répertoires

- Trouvez dans le manuel ce que réalise la commande :

```
ls -lh
```

- Observons ensemble le résultat...

# Utilisateurs

- Chaque utilisateur se connecte sur une machine UNIX avec un indentifiant (*user*)
- Un utilisateur est décrit par :
  - Nom d'utilisateur
  - Mot de passe
  - Identifiant numérique (**UID**)
  - Groupe (primaire)
  - Répertoire *home*
  - Description
  - Shell par défaut

# Super-utilisateur (root)

- Il existe un super-utilisateur : `root`
- Son `UID` est 0
- Il possède tous les droits

Accès au mot de passe root = **faille de sécurité majeure !**

Les utilisateurs peuvent faire partie d'un ou plusieurs **groupes** :

- Un groupe **primaire**
- Optionnellement, des groupes **supplémentaires**

Ceci permet par exemple de :

- Restreindre l'accès à un répertoire ou des fichiers à un seul groupe
- Donner accès à un périphérique uniquement aux utilisateurs appartenant à un groupe spécifique

## Exercice : utilisateurs et groupes (1/2)

1. A l'aide de la commande `whoami`, obtenez votre login
2. A l'aide de la commande `groups` obtenez les groupes auxquels vous appartenez
3. Retrouvez ces information avec la commande `id`



## Exercice : utilisateurs et groupes (2/2)

La commande **cat** permet d'afficher un ou plusieurs fichiers à l'écran (concaténation, voir manuel avec **man cat**) :

1. Utilisez **cat** sur le fichier **/etc/passwd** et retrouvez vos informations utilisateur
2. Utilisez **cat** sur le fichier **/etc/group** et retrouvez les informations de vos groupes

# Permissions

user	group	other
<b><code>rwX</code></b>	<code>r - X</code>	<code>r - -</code>

read  
write  
execute

# Permissions : effets sur un fichier

Effet des permissions sur un **fichier** :

<b>Read</b>	Lit le contenu du fichier
-------------	---------------------------

---

<b>Write</b>	Modifie le contenu du fichier <i>Warning</i> en cas de suppression
--------------	---

---

<b>eXecute</b>	Exécute le fichier (exécutable binaire ou script)
----------------	---

# Permissions : effets sur un répertoire

Effet des permissions sur un **répertoire** :

**Read**

Liste les **noms** (mais pas les métadonnées) des fichiers se trouvant dans le répertoire

---

**Write**

Crée, renomme ou détruit les fichiers se trouvant dans le répertoire

---

**eXecute**

“Ouvre” le répertoire, voit les métadonnées, accède au contenu des fichiers (mais pas aux noms), exécute les fichiers exécutables

# Changement des permissions : chmod (1/2)

user	group	other		
rwX	r - X	r - -	read	= 4
			write	= 2
			execute	= 1

rwX	r - X	r - -	=>	754
421	401	400		
7	5	4		

La commande **chmod** permet de changer les droits des fichiers :

```
$ chmod 754 myfile.c      # Syntaxe numérique (octal)
```

# Changement des permissions : chmod (2/2)

**chmod** accepte également une syntaxe symbolique, avec le format **[u,g,o,a] [+,-,=] [r,w,x]** :

Cible	Modification	Droits
<b>u</b> : user	<b>=</b> : remplacement des droits	<b>r</b> : read
<b>g</b> : group	<b>+</b> : ajout des droits	<b>w</b> : write
<b>o</b> : others	<b>-</b> : suppression des droits	<b>x</b> : execution
<b>a</b> : all		

# Changement des permissions : exemples

```
$ chmod u+x f.c    # ajoute le droit d'exécution pour l'utilisateur  
$ chmod ug=rx f.c  # fixe les droits de l'utilisateur et du groupe à 5 (r-x)  
$ chmod a-x f.c    # retire les droits d'exécution à tout le monde
```

```
# Ces deux exemples produisent le même résultat que : chmod 754 myfile.c  
$ chmod u+rwx,g+rx-w,o+r-xw myfile.c  
$ chmod u=rwx,g=rx,o=r myfile.c
```

# Exercice : permissions

1. Regardez le contenu du fichier `bird00.jpg`
2. Retirez les droits de lecture utilisateur du fichier `images/bird00.jpg`
3. Essayez de regarder le contenu du fichier `bird00.jpg`
4. Retirez les droits d'écriture utilisateur du fichier `images/bird00.jpg`
5. Vérifiez les changements de droit effectués
6. Retirez les droits d'exécution du répertoire `images`
7. Listez le contenu du répertoire `images` avec `ls -l`
8. Redonnez les droits d'exécution au répertoire



Créer et supprimer des  
fichiers/répertoires

---

# Création et déplacement

**touch *file***

Crée le fichier vide *file* si il n'existe pas, sinon met à jour sa date de modification

---

**mkdir *dir***

Crée le répertoire *dir*

---

**mv *source [sources] dest***

Déplace les sources vers la destination *dest* si elle est un répertoire, sinon renomme le fichier *source* en *dest* (possible changement de répertoire)

---

**cp *source [sources] dest***

Copie les sources vers la destination *dest* (*dest* peut être un fichier ou un répertoire)

# Exercice : création et déplacements

1. Créez le répertoire `backup`
2. Copiez l'image `images/bird01.jpg` vers `backup/mybird.jpg`
3. Essayez de copier l'image `images/bird00.jpg`
4. Copiez toutes les images de chien vers `backup`
5. Créez un fichier image vide dans `backup/myimage.png`
6. Renommer le fichier `backup/myimage.jpg`
7. Déplacez **en une commande** les images de chats dont le nom contient les nombres `00` à `09` et `90` à `99` vers le répertoire `backup`

# Suppression

`rm file`      Supprime le fichier *file*

---

`rmdir dir`      Supprime le répertoire *dir*  
Attention, celui-ci doit être **vide** !

## Exercice : suppression (1/2)

1. Tentez de supprimer le répertoire `backup`, pourquoi n'est-ce pas possible?
2. Supprimez le fichier `backup/mybird.jpg`
3. Supprimez tout le contenu du répertoire `backup` (sans supprimer le répertoire) en une commande
4. Supprimez le répertoire `backup`

## Exercice : suppression (2/2)

1. Essayez de supprimer le fichier `images/bird00.jpg`. Refusez de supprimer le fichier. Que s'est-il passé, pourquoi ?
  - ce fichier est donc supprimable, comment faire pour qu'il ne le soit plus ? Testez la solution, puis remettez les droits à ceux d'origine
2. Copiez le répertoire `images` vers `images-backup`, que ce passe-t-il ? Que veut dire l'option proposée ?
3. Effectuez la copie en utilisant l'option adéquate
4. En regardant le manuel de la commande `rm`, supprimez le répertoire `images-backup` et son contenu en une commande

# Le Cauchemar

```
$ rm -rf ~/*.txt  
$ rm -rf ~/* .txt
```



# Éditer un fichier sur une interface texte

Plusieurs éditeurs “textes” sont disponibles, on peut mentionner :

<b>nano</b>	Éditeur léger, le plus facile à appréhender, les raccourcis clavier sont indiqués en bas
<b>vi</b>	Éditeur historique, disponible sur tous les systèmes UNIX, forte courbe d'apprentissage
<b>vim / neovim</b>	Nouvelles versions de vi, beaucoup plus puissant et avec de nombreux plugins disponibles
<b>emacs</b>	Éditeur historique, forte courbe d'apprentissage, extrêmement puissant et bien plus qu'un éditeur, <a href="#"><u>il fait même le café</u></a>



## Exercice : éditer un fichier

- Créez un nouveau fichier contenant la liste de vos courses, en utilisant un des éditeurs mentionnés à la slide précédente
- Vérifiez le contenu du fichier en utilisant une commande déjà vue (autre qu'un éditeur)

# Redirection de flux et pipes

---

# Flux de sortie standard

- Le flux de sortie standard (appelé **stdout**) d'un programme est dirigé par défaut sur le terminal courant
- La commande **echo** affiche les paramètres qui lui sont passés en argument sur **stdout**

```
$ echo ce cours est vraiment génial
ce cours est vraiment génial

$ echo "ce cours est vraiment génial"
ce cours est vraiment génial
```

- Ce flux peut être redirigé vers un fichier avec le caractère **>** :

```
$ echo ce cours est vraiment génial > /tmp/qualite_du_cours.txt
```

## Exercice : flux de sortie

1. Stockez la liste des fichiers du répertoire `images`, ainsi que leurs attributs, dans un fichier nommé `result`
2. Copiez un fichier vers le fichier `result` sans utiliser la commande `cp`
3. Inspectez le contenu de `result`
4. Stockez la liste des fichiers du répertoire `images`, ainsi que leurs attributs, dans un fichier nommé `result`
5. **En utilisant `>>` au lieu de `>`**, copiez un fichier vers le fichier `result` sans utiliser la commande `cp`
6. Inspectez le contenu de `result`, que concluez-vous ?

# Flux d'entrée standard (1/2)

- Le flux d'entrée standard (appelé **stdin**) d'un programme est par défaut associé au clavier du terminal
- La plupart des commandes nécessitant une entrée en paramètre (p.ex. un nom de fichier) utilisent l'entrée standard si aucun argument n'est spécifié

```
$ cat                # attend et répète les entrées utilisateur
$ cat > livre.txt    # concatène tout ce qui est tapé dans
                    # livre.txt => un nouvel EDITEUR de texte !
```

## Flux d'entrée standard (2/2)

- Un fichier peut-être redirigé vers ce flux d'entrée en utilisant le caractère `<` :

```
$ cat < livre.txt    # l'entrée standard est le fichier livre
$ cat livre.txt      # le programme lit depuis le fichier livre,
                    # pas depuis l'entrée standard !
```

- Dans les deux cas ci-dessus, le **résultat est le même**, mais le **comportement est complètement différent !**
- **Ctrl+D** permet d'interrompre le flux d'entrée

# Flux d'erreur standard

- Les messages d'erreur des programmes ne sont en général pas redirigés vers **stdout**, mais vers un flux d'erreur nommé **stderr**
- Par défaut le flux d'erreur **stderr** est redirigé vers le terminal
- Ce flux peut être redirigé vers un fichier avec **2>** :

```
# produit un message d'erreur sur le terminal et un fichier vide
$ ls non_existant_folder > /tmp/error.txt

# produit un fichier contenant le message d'erreur
$ ls non_existant_folder 2> /tmp/error.txt
```

# Exercice : flux d'erreur

1. Utilisez la commande `ls -R /tmp` pour lister récursivement le contenu de `/tmp`
2. Utilisez la même commande, mais redirigez **stdout** dans un fichier
3. Utilisez la même commande, mais redirigez **stderr** dans un autre fichier
4. Enfin, utilisez la même commande, mais redirigez **stdout** et **stderr** vers deux fichiers différents
5. Observez les contenus de ces fichiers



# Flux Standard dans les langages de programmation

	C	C++	Java	Python	Go
<b>Entrée</b>	stdin	std::cin	System.in	sys.stdin	os.Stdin
<b>Sortie</b>	stdout	std::cout	System.out	sys.stdout	os.Stdout
<b>Erreur</b>	stderr	std::cerr	System.err	sys.stderr	os.Stderr

Exemples :

```
printf("Hello world!\n");           // $ ./HelloWorld > hello.txt
```

```
fprintf(stderr, "Some error!\n");   // $ ./Error 2> err.txt
```

```
System.out.println("Hello world!"); // $ java HelloWorld > hello.txt
```

# Pipes

- Les **pipes** (“tubes” en français) permettent de rediriger la sortie standard d’un programme vers l’entrée standard d’un autre
- Dans un shell, on utilise le symbole **|** entre deux commandes pour les utiliser :

```
$ less file          # affiche le contenu du fichier file avec scrolling  
$ ls -R /tmp | less  # affiche la sortie de ls AVEC scrolling
```

Combinés à la philosophie UNIX, les pipes sont au coeur de la **puissance** des interfaces en ligne de commande

**grep *pattern fichier***

Cherche et affiche sur la sortie standard (**stdout**) les lignes du fichier qui contiennent un pattern (e.g. un mot)

---

**grep *pattern***

Cherche le pattern dans l'entrée standard (**stdin**) et affiche sur la sortie standard (**stdout**) les lignes le contenant

Exemple :

```
$ ls -l /dev/ | grep rand
crw-rw-rw-  1 root root      1,   8 mar  3 09:59 random
crw-rw-rw-  1 root root      1,   9 mar  3 09:59 urandom
```

# Exercice : pipes

Quelques commandes utiles pour cet exercice (voir le manuel) :

```
ls, sort, uniq, grep, head, du
```

1. Utilisez la commande `ls -l` (sans autre argument) et le mécanisme de pipe pour lister les fichiers de chats du répertoire `images`
2. Le répertoire `mails` contient des listes d'adresses. À partir de ces fichiers, obtenez une seule liste des membres d'HEPIA, triée par ordre alphabétique et sans doublons
3. Trouver deux méthodes pour lister les deux fichiers les plus volumineux contenus dans le répertoire `mails`

# Introduction aux scripts

---

# Qu'est ce qu'un script ?

- Un script est un fichier texte contenant une séquence de commandes shell
- Un script peut être exécuté comme une commande ou un programme

# Exécuter un script

Un moyen simple pour exécuter un script :

1. Ajouter le bit d'exécution au fichier script
2. L'exécuter

```
$ chmod +x script.sh
```

```
$ ./script.sh
```

# Arguments

Il est possible de récupérer les arguments passés à un script en utilisant des **variables prédéfinies** :

<b>\$0</b>	Contient le nom du script tel qu'il a été appelé (ex. /home/jane/script.sh ou ./script.sh)
<b>\$1, \$2, \$3, etc.</b>	<b>\$1</b> indique le premier argument passé au script, <b>\$2</b> le deuxième, etc.
<b>\$@</b>	La liste des arguments (sans le nom du script)
<b>\$#</b>	Le nombre d'arguments passés au script



## Exercice : premiers scripts

Reprenez ce que vous avez réalisé dans l'exercice précédent pour créer les scripts suivants :

<b>list.sh</b>	Prend un <i>pattern</i> en argument ; affiche sur <b>stdout</b> tous les fichiers (avec leurs méta-données) contenant <i>pattern</i>
----------------	--

---

<b>mails.sh</b>	Prend un ou plusieurs fichiers en argument ; à partir de ce(s) fichier(s), affiche sur <b>stdout</b> la liste des emails des membres d'HEPIA, triée par ordre alphabétique et sans doublons
-----------------	---

---

<b>largest.sh</b>	Prend un répertoire en argument ; affiche sur <b>stdout</b> les deux fichiers les plus volumineux du répertoire, avec leurs métadonnées
-------------------	---

# Systemes de fichiers

---

Manière d'organiser les données sur un support physique (disque dur, clé USB, etc.) :

- Fichiers
- Répertoires
- Nom
- Permissions
- Données
- Metadonnées

Caractéristiques d'un système de fichiers :

- Efficacité en espace
- Efficacité en temps d'accès
- Manière de stocker les données
- Maintien de l'intégrité
- Limitations
- ...

# Types de systèmes de fichiers

Il existe de nombreux types de systèmes de fichiers, par exemple :

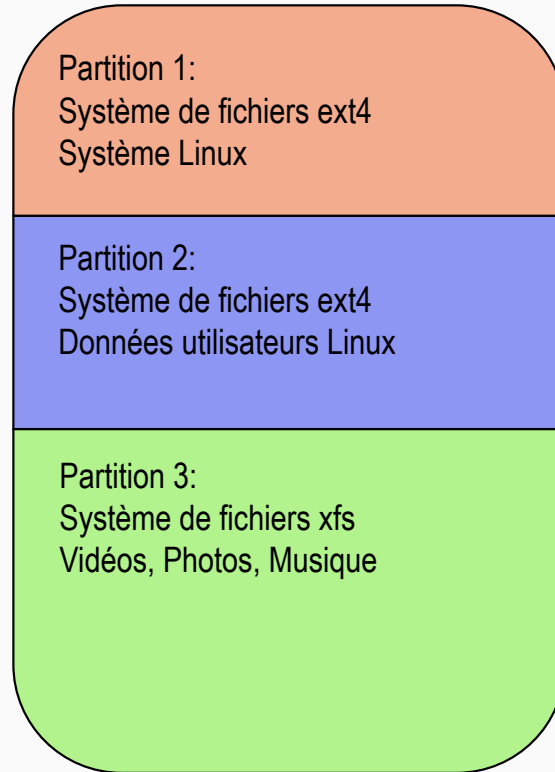
<b>FAT32</b>	Ancien système de fichiers de DOS/Windows, très utilisé sur les clé-usb, lecteurs MP3, etc.
<b>NTFS</b>	Système de fichiers Windows
<b>ext4</b>	Système de fichiers le plus populaire dans GNU/Linux
<b>btrfs</b>	Système de fichiers puissant et avancé pour GNU/Linux
<b>ISO9660</b>	Système de fichiers des CD-ROMs
<b>APFS</b>	Système de fichier d'OSX
<b>NFS</b>	Système de fichiers réseau pour UNIX
<b>IPFS</b>	Système de fichiers distribué et décentralisé, à l'échelle planétaire

# Arborescence (1/3)

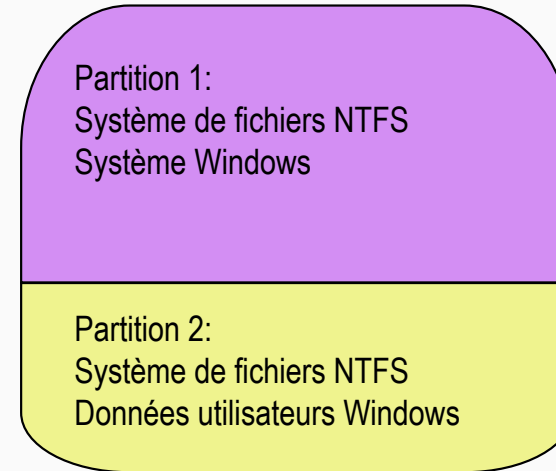
- Tous les systèmes de fichiers sont “montés” (~“greffés”) sur **une seule et même arborescence**
- Par défaut, seul **root** possède les droits pour monter/démonter des systèmes de fichiers
- La plupart des installations GNU/Linux montent automatiquement les disques externes (CD, DVD, clé/disque usb)

# Arborescence (2/3)

## Disque 1

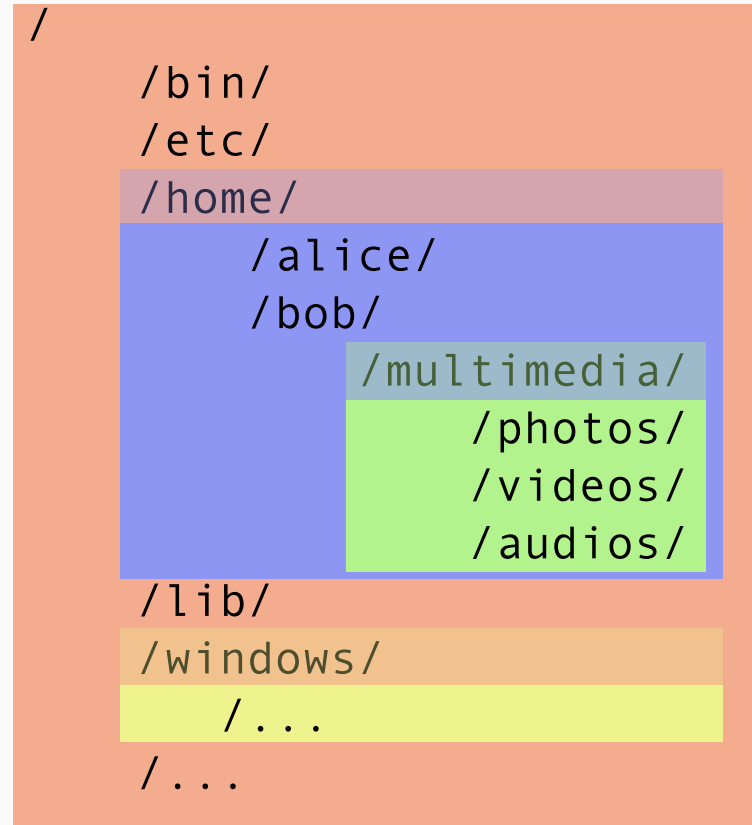


## Disque 2

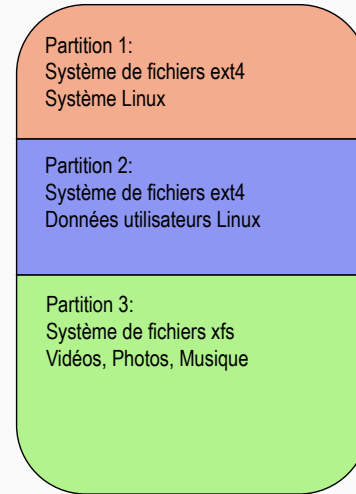


Partitionnement des espaces physiques (disques) en espaces logiques (partitions)

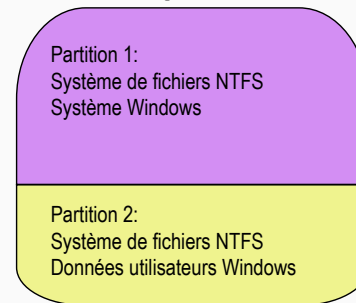
# Arborescence (3/3)



## Disque 1



## Disque 2



Montage des systèmes de fichiers dans l'arborescence

# Arborescence UNIX standard

<b>/bin</b>	Principaux exécutables
<b>/boot</b>	Fichiers de démarrage
<b>/dev</b>	Périphériques
<b>/etc</b>	Configuration système
<b>/home</b>	Données utilisateurs
<b>/lib</b>	Librairies système
<b>/media</b>	Montage des périphériques de stockage (dynamique)
<b>/mnt</b>	Point de montage manuel (statique)
<b>/proc</b>	Informations liées aux processus
<b>/root</b>	Répertoire de l'utilisateur root
<b>/sbin</b>	Exécutables système
<b>/sys</b>	Informations noyau liées aux bus et modules
<b>/usr</b>	Applications et librairies utilisateurs
<b>/var</b>	Données qui grandissent (logs, cache, mail, ...)
<b>/tmp</b>	Fichiers temporaires



# *Everything is a file*

- Plusieurs concepts sont représentés par des fichiers **virtuels**
- Ces fichiers ne sont pas stockés sur disque, mais sont une **vue des ressources noyau**
- Exemples :
  - les périphériques sont visibles dans `/dev`
  - les processus (et certaines info système) sont visibles dans `/proc`
  - les informations sur les bus et modules noyau dans `/sys`

# Exemples d'utilisation de /dev

Créer une image ISO à partir d'un CD-ROM :

```
$ cat /dev/cdrom > image.iso
```

Remplir un disque avec des données pseudo-aléatoires :

```
$ dd if=/dev/urandom of=/dev/sdb1
```

Afficher un message dans le terminal courant :

```
$ tty  
/dev/pts/8  
$ echo "hello world" > /dev/pts/8
```

# Exercice : systèmes de fichiers

1. Grâce à la commande **mount**, trouvez où sont montés sur votre OS les systèmes de fichiers de type ext4, proc et devtmpfs
2. Sachant qu'un périphérique peut être de type bloc ou caractère et que cette information est indiquée par "b" ou "c" dans le mode du fichier, listez tous les périphériques de type bloc
3. Grâce au périphérique **zero** (et dd), remplissez un fichier de 1MB de zéros
  - **attention : la commande peut donner lieu à de GROS fichiers si mal utilisée !**
4. Grâce au périphérique **null**, listez récursivement, sans être root, le contenu de **/tmp**, **sans** que le terminal n'affiche de sortie d'erreur

# Processus



# Qu'est-ce qu'un processus ?

**Processus = instance d'un programme en cours d'exécution**

- Un processeur (ou coeur) ne peut **exécuter qu'un seul processus à la fois !**
- Le système d'exploitation alterne, très fréquemment dans le temps, l'exécution de plusieurs processus sur un même processeur
  - donne l'**illusion** que plusieurs processus s'exécutent "en même temps"
  - technique appelée *multi-tasking*

# Anatomie d'un processus

Un processus possède :

- Son code en langage machine
- Des segments de mémoire (code, données, pile, etc.)
- Des descripteurs de fichiers (sortie et entrée standard, fichiers ouverts, etc.)
- Un propriétaire et un ensemble de permissions
- Un état à un moment donné, aussi appelé “contexte” (états des registres processeur, etc.)

# Processus UNIX

- La commande **ps** permet de lister les processus
- Dans UNIX, tous les processus ont :
  - un identifiant numérique (PID)
  - un processus parent (sauf le processus initial 0)

# Monitoring du CPU et de la RAM

La commande **top** affiche l'utilisation CPU et RAM du système :

```
top - 19:02:23 up 142 days, 8:52, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 1184 total, 1 running, 1183 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 515811.3 total, 93546.3 free, 12074.3 used, 410190.7 buff/cache
MiB Swap: 8192.0 total, 8155.5 free, 36.5 used. 499973.8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1387277	root	20	0	11724	5352	3440	R	1.6	0.0	0:00.24	top
3910353	nexus	20	0	7462412	3.0g	28072	S	1.3	0.6	646:13.46	qemu-sy+
798	root	25	5	0	0	0	S	1.0	0.0	896:32.38	ksmd
195015	nexus	20	0	8411112	4.0g	28824	S	1.0	0.8	651:52.35	qemu-sy+
3910355	nexus	20	0	7379656	3.0g	28108	S	1.0	0.6	643:31.16	qemu-sy+
2057	root	20	0	83008	3508	2912	S	0.3	0.0	200:02.87	irqbala+
1065265	root	20	0	0	0	0	I	0.3	0.0	2:45.41	kworker+
1	root	20	0	166604	12136	8488	S	0.0	0.0	2:15.71	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:08.02	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par+
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	slub_fl+
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker+
11	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_perc+
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tas+
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tas+



## Exercice : ps

1. Tapez la commande **ps**. Qu'est ce que les processus affichés ont en commun ?
2. Tapez la commande **ps -ef** (voir man). Trouvez un processus qui est l'enfant d'un autre.
3. Vérifiez avec la commande **pstree** que c'est bien le cas. Quel est le PID du processus initial (le plus ancien parent de tous les processus) ?
4. Trouvez un moyen de lister tous les processus vous appartenant.

# Signaux



# Qu'est-ce qu'un signal ?

- Les signaux permettent une communication limitée entre les processus
- A chaque signal est associé une action par défaut :
  - **Term** : termine le processus
  - **Ign** : ignore le signal
  - **Core** : termine le processus et crée un fichier *core dump*<sup>1</sup>
  - **Stop** : stoppe le processus
  - **Cont** : continue le processus s'il est stoppé
- La commande **kill -l** affiche les signaux disponibles avec leurs noms et numéros
- **man 7 signal** liste les signaux standards, leurs noms, actions et numéros

---

<sup>1</sup>Voir le manuel avec **man core**

# Signaux utiles

Nom	Action	Description	Numéro
<b>SIGINT</b>	Term	Interrompt le processus	2
<b>SIGKILL</b>	Term	Tue le processus immédiatement	9
<b>SIGTERM</b>	Term	Termine le processus (propre)	15
<b>SIGTSTP</b>	Stop	Suspend l'exécution du processus	20
<b>SIGCONT</b>	Cont	Continue le processus suspendu	18
<b>SIGUSR1</b>	Term	Signal utilisateur 1	10
<b>SIGUSR2</b>	Term	Signal utilisateur 2	12

# Envoyer un signal

La commande **kill** envoie un signal à un processus :

```
# signal peut être spécifié par son nom ou numéro  
kill -signal [pids]
```

Exemples :

```
$ kill -SIGKILL 2111 2120  
$ kill -9 2111
```

- Tout processus peut changer l'action par défaut d'un signal
  - **sauf** pour les signaux **SIGKILL** et **SIGSTOP** !
- Exemples :
  - sauver l'état d'un processus sur disque avant de quitter lorsque SIGTERM est reçu
  - redémarrer le processus lorsque SIGINT est reçu

# Signaux depuis le shell

Les raccourcis claviers suivants permettent d'envoyer un signal au processus s'exécutant dans le terminal courant :

**Ctrl+C**    Envoie **SIGINT**

---

**Ctrl+Z**    Envoie **SIGTSTP**

Attention **Ctrl+D** n'est pas un signal, mais envoie End-Of-File (EOF) sur l'entrée standard !

# Exercice : signaux

1. Exécutez l'éditeur de texte graphique `gedit` depuis le terminal
2. Interrompez-le en envoyant `SIGINT` de deux manières différentes
3. Exécutez un éditeur de texte dans le shell (i.e. non graphique) et tapez du texte **sans sauvegarder**
4. Depuis un autre terminal, terminez le programme en envoyant `SIGTERM`
5. Recommencez, mais cette fois-ci interrompez-le en envoyant `SIGINT`
6. Finalement, terminez le programme en envoyant `SIGKILL`
  - quelles sont les différences de comportement ?



# Jobs



# Processus en avant et arrière plan

- Par défaut, tout programme exécuté depuis le shell s'exécute en avant-plan (*foreground*)
- Un processus en avant-plan bloque le shell tant qu'il n'est pas terminé
- Un programme peut être exécuté en arrière-plan (*background*) en ajoutant **&** après la commande :

```
$ xcalc&  
[1] 271802  
$ xterm&  
[2] 271803
```

# Commande jobs

- La commande **jobs** (interne au shell) affiche les jobs du shell courant où le numéro entre crochets [ ] indique le numéro du job :

```
$ jobs
[1]-  Running                  xcalc &
[2]+  Running                  xterm &
```

- Le caractère **+** indique le job courant (attention : courant  $\neq$  avant-plan)
- **Ctrl+Z** stoppe le processus en avant-plan
  - envoie le signal SIGTSTP au processus

# Contrôle de jobs

- La commande **bg** (interne au shell) continue l'exécution d'un processus stoppé en arrière-plan
  - envoie le signal SIGCONT au processus
- La commande **fg** (interne au shell) agit comme **bg**, mais place le processus en avant-plan
- Un numéro de job peut être spécifié en argument à **bg** ou **fg**
  - si aucun argument n'est spécifié, la commande porte sur le job courant

# Exercice : jobs

1. Exécutez l'éditeur de texte `gedit` en arrière-plan
2. Dans le même terminal, exécutez le programme `xeyes`
3. Stoppez `xeyes` avec Ctrl+Z
4. Placez `gedit` en avant-plan, stoppez-le, testez sa réactivité, puis continuez son exécution en arrière-plan
5. Toujours dans le même terminal, éditez un fichier en utilisant `vim`
6. Stoppez-le avec Ctrl+Z
7. Continuez l'exécution en arrière-plan de `xeyes`
8. Continuez l'exécution en avant-plan de `vim`
9. Sortez de `vim`
10. En manipulant les jobs et avec Ctrl+C, terminez `xeyes` et `gedit`

# Connexions a distance: SSH (Secure Shell)

---

- Travail à distance, comme si on était sur un terminal de la machine (local)
- Accéder à une machine qui ne possède pas de terminal
- Transférer des fichiers de manière sécurisée
- Connexion sécurisée pour des clients X distants (pour sur-simplifier: interface graphique distante)

# Utilisation

- Se connecter avec la commande **ssh** selon la syntaxe suivante :

```
$ ssh login@mon.server.org
```

- Vérifier que le *fingerprint* du serveur correspond bien à celui attendu
- Entrer son mot de passe
- ... réaliser les opérations souhaitées ...
- Terminer la connexion distante en mettant fin au shell associé avec **exit** ou **Ctrl+D**



# Exercice : ssh

1. Connectez-vous au serveur 10.136.26.133, dont le *fingerprint* est :

```
SHA256:dMjJfL/CweskwJK6g+L4vtCh0CgYsYYR7R8aoHISA2U
```

2. Listez le contenu de votre répertoire *home* sur ce serveur
3. En utilisant un éditeur de texte, créez et éditez un nouveau fichier dans un répertoire nommé **votre\_nom-prenom**
4. Terminez la connexion SSH

- La commande **scp** permet de copier des fichiers d'un serveur distant vers la machine locale et vice-versa :

```
$ scp login@mon.server.org:/path/to/my/file /path/to/destination
```


```
$ scp /path/to/my/file login@mon.server.org:/path/to/destination
```

- Les formats de chemin habituels sont disponibles (\*, ?, ~, etc.)

## Exercice : scp

1. Copiez le fichier du répertoire distant `votre_nom-prenom` sur votre machine locale
2. Modifiez le fichier copié
3. Copiez ce fichier modifié dans le répertoire distant `votre_nom-prenom`

# Authentification par clé publique : principe

- Vous possédez une **clé publique** et une **clé privée**
- La **clé privée** VOUS identifie sur UNE MACHINE (une clé par utilisateur et par machine)
- La **clé publique** est associée à la **clé privée** et elle sera distribuée à toute entité avec laquelle vous voulez communiquer
- La **clé publique** est utilisée pour chiffrer un message qui vous est envoyé
  - **seul la clé privée permet de le décoder**
- Il n'est donc pas dangereux de distribuer sa **clé publique**
-  La **clé privée** est **personnelle** et ne doit **jamais quitter la machine !**

# Authentification par clé publique : protocole

- **Client** Bonjour je me présente, voici ma clé publique
- **Serveur** OK, elle fait bien partie de la liste des clés que je connais (il faudra donc fournir sa clé au serveur avant authentification)
- **Serveur** Voici un message chiffré par votre clé publique, serez-vous capable de le déchiffrer ?
- **Client** Oui je peux, j'ai la clé privée et voici le message déchiffré
- **Serveur** OK, je confirme que vous avez la bonne clé privée, car le message est bien déchiffré, vous êtes authentifié

# Authentification par clé publique : pour et contre

Utilisée par presque tous les services en ligne et APIs (github, gitlab, etc.)

## Avantages

- Facilité de connexion et automatisation (pas de mot de passe)
- Plus sécurisé qu'un mot de passe, car aucune information secrète n'est transférée (et souvent les mots de passe choisi sont faibles)

## Inconvénients

- Parfois difficile de gérer un grands nombre de clés publiques/privées
- Demande quelques connaissances techniques

# Authentification par clé publique : en pratique

1. Créer une paire de clé publique/privée (de préférence avec passphrase) :

```
$ ssh-keygen
```

2. Copier la clé publique sur le serveur :

```
$ ssh-copy-id login@mon.server.org
```

3. Se connecter au serveur :

```
$ ssh login@mon.server.org
```